DERWENT-ACC-NO: 2003-419180

DERWENT-WEEK: 200339

TITLE: Memory region swapping method for system area network, involves swapping out memory region if the current number of outstanding operations to the memory region is zero

INVENTOR: CRADDOCK, D F; GREGG, T A ; RECIO, R J ; SCHMIDT, D W

PATENT-ASSIGNEE: INT BUSINESS MACHINES CORP[IBMC]

PRIORITY-DATA: 2001US-0942632 (August 30, 2001)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 20030046505 A1	March 6, 2003	N/A	024	G06F 013/00

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
US20030046505A1	N/A	2001US-0942632	August 30, 2001

INT-CL (IPC): G06F013/00

ABSTRACTED-PUB-NO: US20030046505A

BASIC-ABSTRACT:

NOVELTY - The method involves instructing a process, which includes setting a quiescent indicator to the memory region, to inhibit further operations on a memory region. If the current number of outstanding operations to the memory region is zero, that region is swapped out.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:

(a) a memory region swapping computer program; and

(b) an apparatus for swapping out a memory region in a system area network

USE - For system area network.

ADVANTAGE - Enables reusing of all or part of the memory that makes up a memory region by another application during the period it is not being used by the owning application. Provides guarantees that input-output operations will not be made to a memory region that is being swapped out.

DESCRIPTION OF DRAWING(S) - The figure shows the diagram of the distributed computer system.

CHOSEN-DRAWING: Dwg.1/13

TITLE-TERMS: MEMORY REGION METHOD SYSTEM AREA NETWORK MEMORY REGION CURRENT NUMBER OUTSTANDING OPERATE MEMORY REGION ZERO

DERWENT-CLASS: T01 W01

EPI-CODES: T01-F05E; T01-S03; W01-A06E2B;

SECONDARY-ACC-NO:
Non-CPI Secondary Accession Numbers: N2003-334567

US-PAT-NO: 6615340

DOCUMENT-IDENTIFIER: US 6615340 B1

TITLE: Extended operand management indicator structure and method

DATE-ISSUED: September 2, 2003

US-CL-CURRENT: 712/209, 712/217 , 712/218

APPL-NO: 09/ 533650

DATE FILED: March 22, 2000

——— KWIC ———

Detailed Description Text - DETX (40):
Many computers include multiple CPUs that operate on a common memory--multiprocessors. Each CPU can be executing different programs or sometimes the same program. The CPUs typically do not share registers. They share the main memory and sometimes some levels of cache memory. So long as they do not update common areas of memory, there should be no discemable effect on the correctness of the results of the programs executing on separate processors. When multiple CPUs undertake to update the same memory areas then careful coordination is required to produce predictable results. It is common programming practice to place a claim on an area of memory by storing an identifier for the thread, process, or CPU into a memory area that acts as a lock for another memory area. The second memory area or areas contain the data protected by the lock. It is not sufficient to merely load a memory lock word to ensure that some other thread has not reserved it. In that instant of time between fetching the current value from memory, comparing it (perhaps for zero meaning free), and storing this thread ID into it, some other CPU could have carried out the same sequence of operations and may have stored a different ID into that lock word. The programs in both CPUs would then continue executing as though they both own the storage protected by the lock word. To avoid this, many processor implementations provide an atomic change instruction. This is often called Compare And Swap. Compare And Swap will compare the value of an operand with an area in shared memory in an interlocked fashion: No other CPU will be able to store into that memory location while the Compare And Swap is accessing it. Compare And Swap is said to be an atomic access to the memory operand. The processor design and/or the software must arrange that the lock word subject of a Compare And Swap instruction is actually accessed from a shared level of memory instead of from private caches where two processors could otherwise reflect two different realities. Use of Compare And Swap for locking is based on the honor system. A rogue or erroneously coded program could easily just begin using a memory area without exercising the locking discipline, which is based on common agreements for sharing.

Detailed Description Text - DETX (41):

In FIG. 12 two different threads are executing programs simultaneously on two different CPUs 1210, 1211. Both programs appear to be the same but could just be common instruction sequences used for the same task: obtaining an exclusive lock on a shared word of memory. An instruction 1201 and an instruction 1205 in the respective CPUs load different values into their respective R1 operands. They each then proceed to Compare And Swap the memory word operand 1204 with zero (meaning free). Only one can succeed. In the example in FIG. 12 an instruction 1202, executing in processor 1210, accesses a lock word 1204 first. Winner instruction 1202 will store its signature from R1 into memory word 1204, fall through a branch 1203, and continue executing from there. A loser 1206 executing in processor 1211 will loop through a branch 1207 back to its Compare And Swap 1206. Right processor 1211 will continue in this tight loop until some other processor, such as processor 1210, restores memory word 1204 to the value needed by 1206 or until an interrupt signal (possibly a timer expiration) interrupts processor 1211. Processor 1211 will loop, `spinning` on spin lock, memory word 1204.